Contiguous memory allocation in Linux user-space

Guy Shattah, Christoph Lameter
Linux Plumbers Conference 2017
Contents

▪ Agenda
  • Existing User-Space Memory Allocation Methods
  • Fragmented Memory vs Contiguous Memory
  • Performance
  • Memory Pinning
  • Proposed Solutions
  • Suggested Implementations
  • Summary
Existing User-space Memory Allocation Methods

- **Existing memory allocation techniques**
  - On the stack: `alloca()` / function stack
  - On the heap: `malloc()`/`calloc()`/`realloc()`
  - Memory mapping: `mmap()`: Anonymous memory or with file descriptor
  - Pre-allocation (i.e. static)

Q: **What do these techniques have in common?**

- **Fragmented memory**
  - Allocated memory resides on separate pages.
  - Virtually: Memory seems to be contiguous to the user.
  - Psychically: Pages are spread over all the ram.

- **Contiguous memory:**
  - Psychically: A sequence of memory pages without holes.
  - Virtually: the same.
  - No existing userspace API allocates contiguous memory.
Is Fragmented Memory Bad for Us?

- **Software Solution:**
  - Virtually mapped contiguous areas.
    - MMU Maps: Virtual Address → physical address
  - In Linux: Demand paging and reclaim.

- **Hardware Solution:**
  - IOMMU serves as MMU for devices
  - DMA can do vector I/O
    - Gather data from fragmented memory blocks
    - Scatter data to fragmented memory blocks
    - Hence DMA scatter/gather

**So why bother?**
Performance Comparison: Memalign vs Contiguous

<table>
<thead>
<tr>
<th>size</th>
<th>memalign()</th>
<th>contiguous pages</th>
<th>improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>8.74</td>
<td>10.65</td>
<td>22%</td>
</tr>
<tr>
<td>4</td>
<td>14.83</td>
<td>23.47</td>
<td>58%</td>
</tr>
<tr>
<td>8</td>
<td>35.64</td>
<td>48.52</td>
<td>36%</td>
</tr>
<tr>
<td>16</td>
<td>76.10</td>
<td>96.02</td>
<td>26%</td>
</tr>
<tr>
<td>32</td>
<td>141.45</td>
<td>195.92</td>
<td>39%</td>
</tr>
<tr>
<td>64</td>
<td>406.52</td>
<td>383.80</td>
<td>-6%</td>
</tr>
<tr>
<td>128</td>
<td>543.58</td>
<td>780.68</td>
<td>44%</td>
</tr>
<tr>
<td>256</td>
<td>1,018.24</td>
<td>1,545.88</td>
<td>52%</td>
</tr>
<tr>
<td>512</td>
<td>2,003.87</td>
<td>3,133.44</td>
<td>56%</td>
</tr>
<tr>
<td>1,024</td>
<td>4,000.76</td>
<td>4,761.60</td>
<td>19%</td>
</tr>
<tr>
<td>2,048</td>
<td>5,837.46</td>
<td>7,216.94</td>
<td>24%</td>
</tr>
<tr>
<td>4,096</td>
<td>7,747.90</td>
<td>9,077.66</td>
<td>17%</td>
</tr>
<tr>
<td>8,192</td>
<td>9,224.03</td>
<td>11,140.40</td>
<td>21%</td>
</tr>
<tr>
<td>16,384</td>
<td>9,109.37</td>
<td>11,561.70</td>
<td>27%</td>
</tr>
<tr>
<td>32,768</td>
<td>9,133.99</td>
<td>11,647.37</td>
<td>28%</td>
</tr>
<tr>
<td>65,536</td>
<td>9,133.65</td>
<td>11,662.10</td>
<td>28%</td>
</tr>
<tr>
<td>131,072</td>
<td>9,179.69</td>
<td>11,694.01</td>
<td>27%</td>
</tr>
<tr>
<td>262,144</td>
<td>9,150.13</td>
<td>11,691.50</td>
<td>28%</td>
</tr>
<tr>
<td>524,288</td>
<td>9,149.54</td>
<td>11,706.13</td>
<td>28%</td>
</tr>
<tr>
<td>1,048,576</td>
<td>9,149.36</td>
<td>11,714.85</td>
<td>28%</td>
</tr>
<tr>
<td>2,097,152</td>
<td>9,161.54</td>
<td>11,715.61</td>
<td>28%</td>
</tr>
<tr>
<td>4,194,304</td>
<td>9,176.27</td>
<td>11,716.65</td>
<td>28%</td>
</tr>
<tr>
<td>8,388,608</td>
<td>9,199.97</td>
<td>11,716.78</td>
<td>27%</td>
</tr>
</tbody>
</table>
Requirement: Contiguous Memory Area

- Contiguous Memory Area
  - Scatter/gather has performance issues.
  - Better cache hits.
Memory Pinning and Compaction Thread

- DMA Operations Typically Require Memory Pinning
  - Memory pinning prevents kernel from:
    - Swapping out memory.
    - Relocation of pages [change mapping of Virtual page -> physical page]
    - Replacing small pages with huge pages (THP)
    - Hot Plug
    - Other compaction thread issues?

- How is memory pinning implemented?
  - Increasing the RefCount on a page struct.
    - Userspace - Using sub-system specific API
    - Kernel - memory registration in the RDMA subsystem.
Huge Pages

Solution #1: Using Huge / Giant Pages

- **Huge Pages:**
  - mmap using the MAP_HUGETLB flag
  - mmap fails if no huge pages are available.
  - Memory continuity is guaranteed (within a single page).
  - Memory has to be divided to several memory pools (4K, 2M, 1G on x64)
  - Requires Memory reservation.

**Advantage:**
- Allocated memory is less fragmented:

**Disadvantage:**
- Pool allocation requires root-user intervention.
- Pool size has to be pre-determined.
- Memory continuity is not guaranteed (over multiple pages).

kernel needs to be built with CONFIG_HUGETLBFS (present under “File systems”) and CONFIG_HUGETLB_PAGE
Solution #2: Allocating Huge Pages On the Fly

- **Transparent Huge Pages (THP):**
  - mmap with a hint from madvice (MADV_HUGEPAGE flag) instructs the kernel to try construction of huge pages on the fly.
  - Fall back to small pages when huge pages not available.

- **Advantages:**
  - Doesn’t require root-user intervention.
  - Doesn't require memory pool reservation.

- **Disadvantage:**
  - THP Allocation isn’t guaranteed to succeed (but there is a fallback).
  - Memory continuity is not guaranteed (over multiple pages).
  - Background CPU work: Consolidate small pages and replace small ones to huge ones.
  - Transparent huge pages are for performance optimization only.
Implementation of Contiguous Memory Allocator in User Space Using Huge Pages

- **Availability:**
  - Huge pages might not be configured on the machine.
  - When configured – pool is limited.

- **Non-continuity:**
  - Allocation of multiple huge pages does not guarantee continuity.
Solution #3: Specific HW support – ARMv8-A

- **Contiguous block entries.**
  - ARMv8-A architecture provides a feature known as contiguous block entries - efficiently uses TLB space.
  - Each TLB entry contains a *contiguous bit*. When set, bit signals to TLB that it can cache a single entry covering translations for multiple blocks.
  - The TLB can cache one entry for a defined range of addresses. Makes it possible to store a larger range of Virtual Addresses within the TLB than is otherwise possible.

- **The contiguous blocks must be adjacent and correspond to a contiguous range of Virtual Addresses.**
  - 16 × 4KB adjacent blocks giving a 64KB entry with 4KB granule.
  - 128 × 16KB adjacent blocks giving a 2MB entry for L3 descriptors when using a 16KB granule.
  - 32 × 64Kb adjacent blocks giving a 2MB entry with a 64KB granule.
  - 32 × 32MB adjacent blocks giving a 1GB entry for L2 descriptors
The Continuous Memory Allocator

Solution #4: Allocating Physically contiguous Memory on Bootup

- CMA: Contiguous Memory Allocator
  - Unavailable for user-mode.
  - Kernel code can request allocation of contiguous memory.
  - CMA requires memory reservation during machine startup.
  - CMA has to be integrated with the DMA subsystem:
    - CMA lets moveable pages use reserved area. Prioritizes clients with special needs.
    - GCMA (Guaranteed CMA) - an improvement over CMA (latency, moving other process’ pages)
      - Uses a client to allocate and vacate memory.

- Issues:
  - Memory reservation technique.
  - Kernel-Space only.

- Would it be possible to implement a user space API for CMA?
The mmap MAP_CONTIG flag

- Flag instructs mmap to allocate contiguous memory
- Idea was first introduced 13 years ago (in IEEE Std 1003.1, 2004 Edition as POSIX_TYPED_MEM_ALLOCATE_CONTIG.)
  - Originally implemented on small number of devices of which most are embedded devices, e.g., BlackBerry.
  - Has since been forgotten.
The mmap MAP_CONTIG flag

▪ Populates all pages tables for a mapping
  ✔ Hence implies MAP_POPULATE

• Anonymous memory allocation:
  \[
  p = \text{mmap}(0, \text{size}, \\
  \text{PROT\_READ|PROT\_WRITE, MAP\_PRIVATE | MAP\_ANONYMOUS | MAP\_CONTIG, -1, 0});
  \]

• Or with a file descriptor:
  \[
  \text{int fd = open("/dev/zero", O\_RDWR);}
  p = \text{mmap}(0, \text{size}, \\
  \text{PROT\_READ|PROT\_WRITE, MAP\_PRIVATE | MAP\_CONTIG, fd, 0});
  \]
The mmap MAP_CONTIG flag – Suggested Implementations

- System tries to use largest available memory blocks to construct contiguous memory area.

- Use case: user requested 3,072Kb (3M)
  - On supported machines, where it is possible to allocate contiguous memory areas larger than a single page:
    - Look for contiguous memory block of 4,096Kb, put the spare 1,024 back in the pool (256 pages, 4Kb each).
  - Divide request:
    - Where larger allocation not supported:
      - i.e.: Allocate 2,048 (one huge page) and than allocate multiple small pages which sum to 1,024.

- Perhaps existing defragmentation passes could attempt to make those areas as contiguous as possible when the pass attempts to constructs THPs?
Summary

▪ **Suggested Implementations**
  • Allocate contiguous areas larger than a single page (on supported architectures).
  • Improve existing defragmentation passes to help make memory areas as contiguous as possible when the pass attempts to constructs THPs.

▪ **Additional Suggestions**
  • Add userspace api for CMA.
  • Add madvise hint to suggest memory should be contiguous.
Q&A

Discussion
Thank You